Developing LPF's Data Management Unit

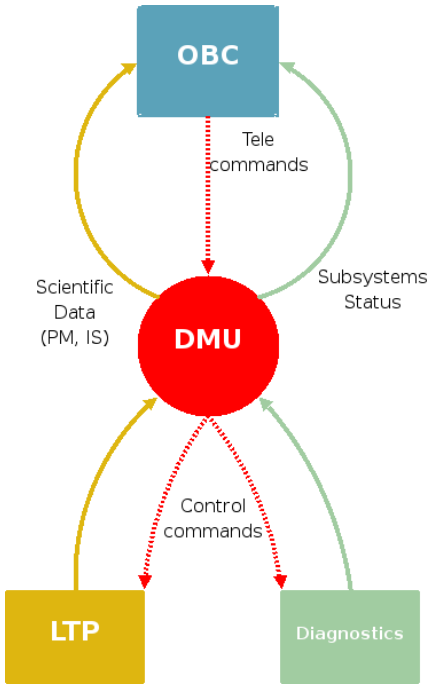
Mission critical software development in LISA Pathfinder

José A Ortega Ruiz

Institut d'Estudis Espacials de Catalunya

6th LISA Symposium

The Data Management Unit



Goals

- M&C of LTP subsystems
- Delivery of science data
- Remote LTP control from OBC/Earth

Challenges

- Robustness/reliability
- Timeliness (real time)
- Small footprint

Hard friends

,

Processor: limited power

- ERC32, RISC based
- 12 MHz, 32 bit buses

FPGA: the world is flat

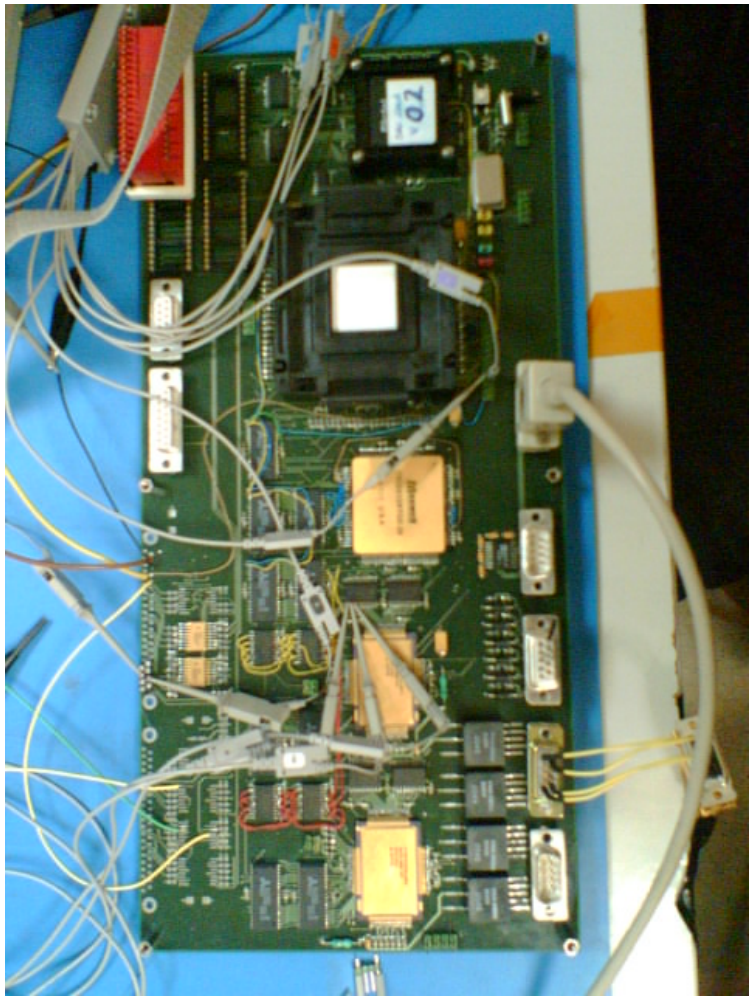
- Memory mapped access to data units. and communications buses.

Storage: secure but scanty

- 1Mb RAM memory, EDAC
- 2Mb EEPROM memory
- 64Kb PROM persistent storage

Communications: military security

- MIL-STD-1553B standard bus/controller
- 16 bits + 4 bit Hamming code



Architecture: fitting and dependable

Problems and solutions

- Fault tolerance → Modifiability/Telecharge
- Chicken & Egg → Split

Boot Software

In charge of booting the system, setting up the communications link with the OBC and managing the application's code.

Application Software

Gets the lion's share of the DMU operation time. Read from the EEPROM, is totally replaceable from Earth.

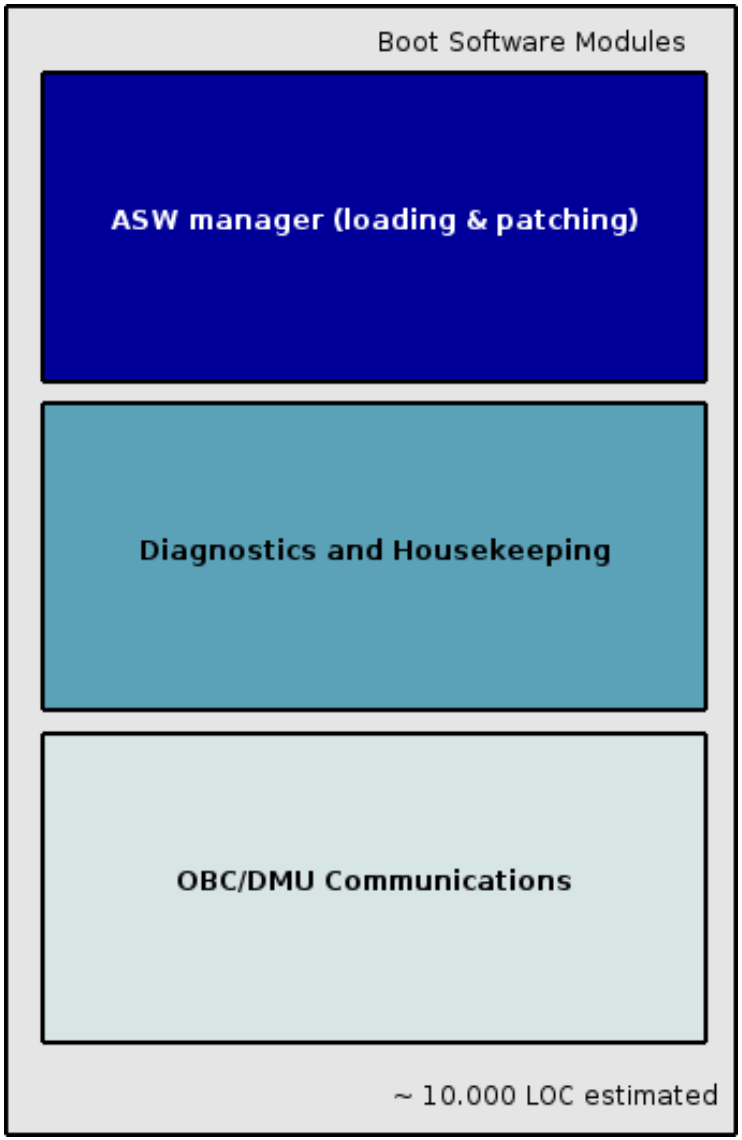
As simple as possible...

Keep it simple and small

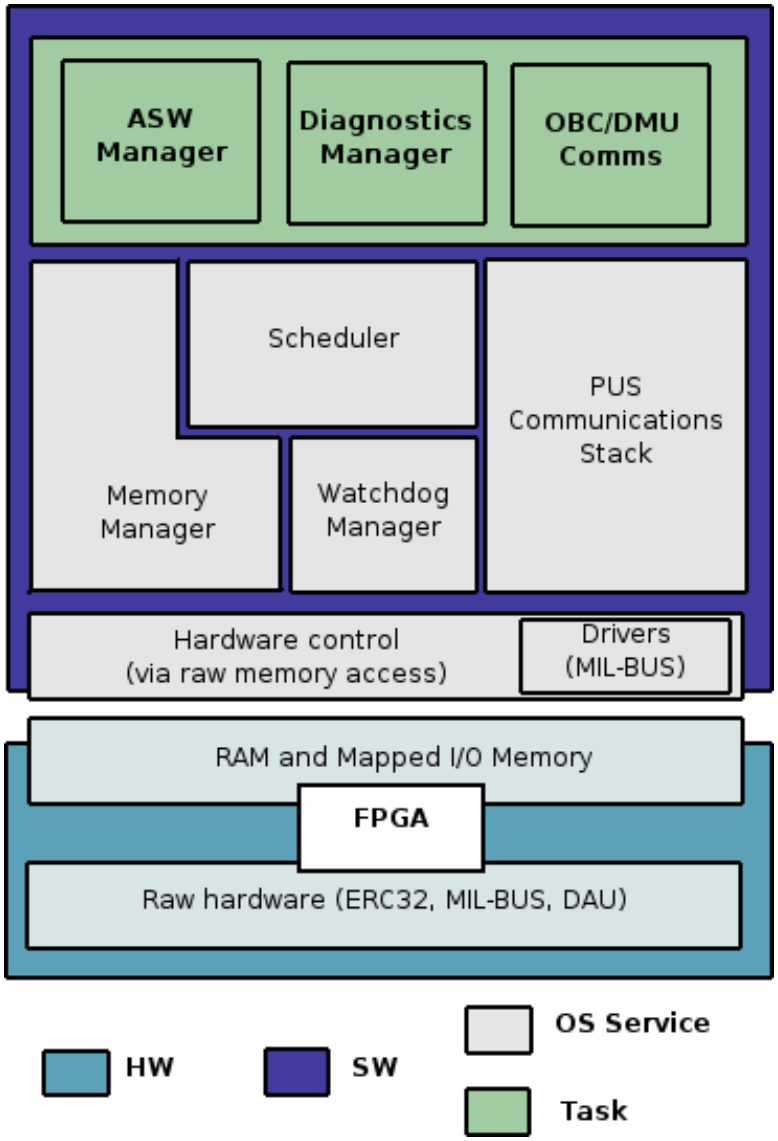
- Hardware limitations (64 Kb PROM)
- Design limitations: KISS
- Thus, minimal footprint and complexity

... but not simpler

- HAL (no room for COTS)
- Communications layer
- ASW Management
- Housekeeping
- Thus, (non-RT) OS-like functions needed

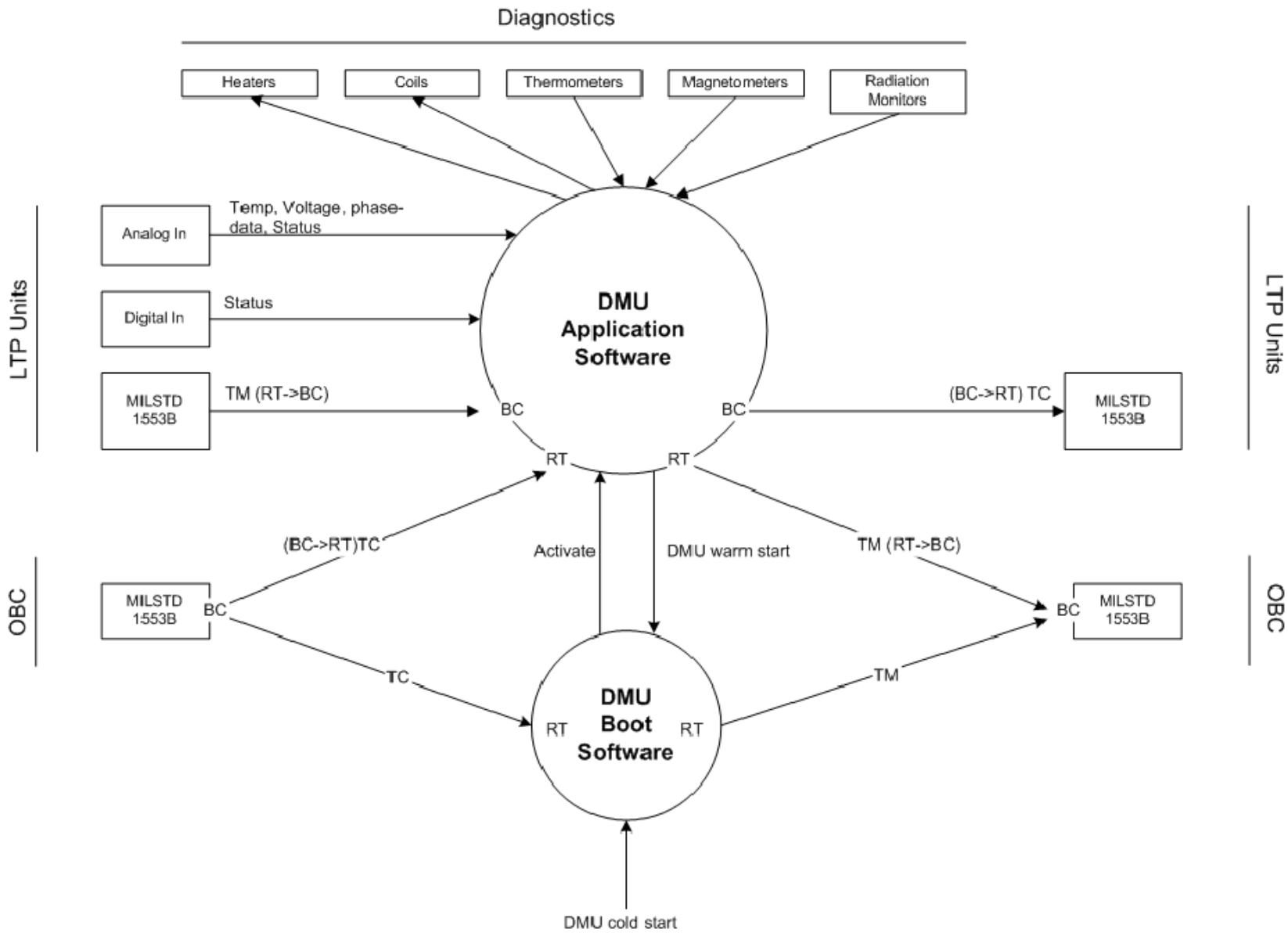


A baby OS



- **FPGA Magic** Uniform access to HW in an almost contiguous memory space. Some addressing quirks.
- **Memory Manager** Completes the world-is-flat memory view and exposes shared state (System Data Pool) to external systems (OBC).
- **Multiple tasks** We use a simple, polling-based scheduler. Asynchronous, interrupt-driven code minimised.
- **Communications stack** on top of the memory-mapped MIL-BUS controller (which uses its own RISC processor), a layered comms protocol: HW, data-link and transport layer using ESA's PUS.
- **If all else fails**, a HW watchdog protects us against deadlocks.
- **High-level tasks** Notably, the ASW Manager handles remote patching of the ASW, the *real* application, stored in (modifiable) EEPROM.
- The BSW life-cycle ends when the ASW Manager loads the ASW into RAM, in an operation analogous to Unix's exec.

The Real (time) Thing

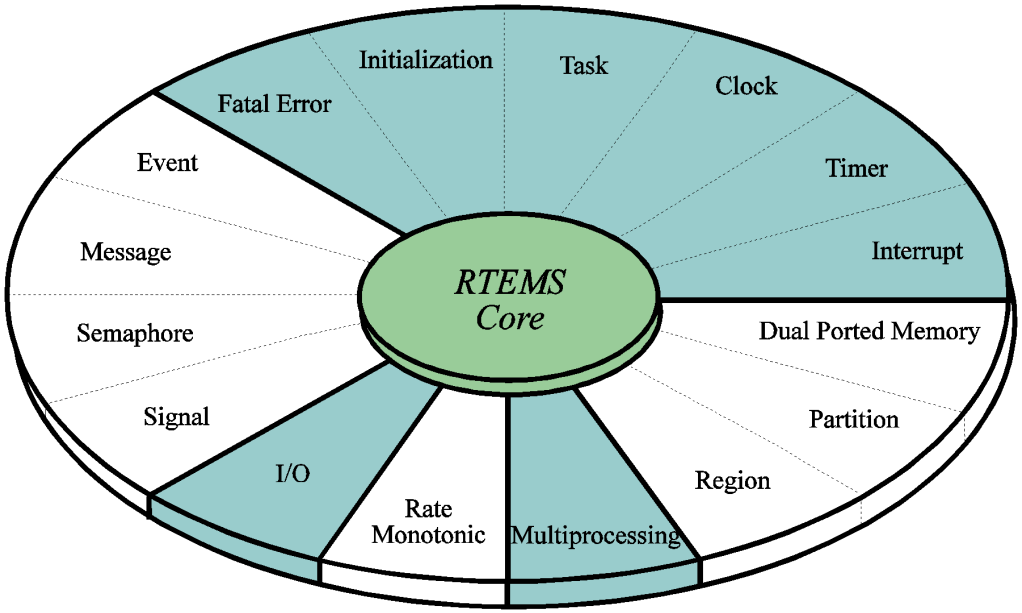


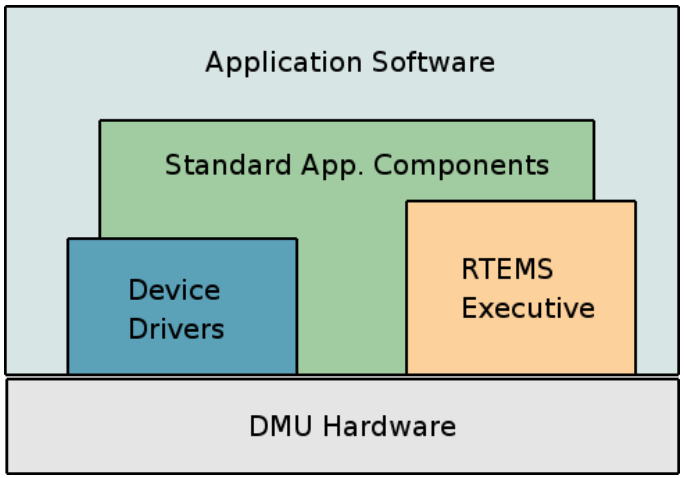
An affordable OS

From the SW development's point of view, the key ASW functionality is data acquisition and post-processing from the Laser/Phasemeter at 100Hz $\Rightarrow$ Real-time requirements

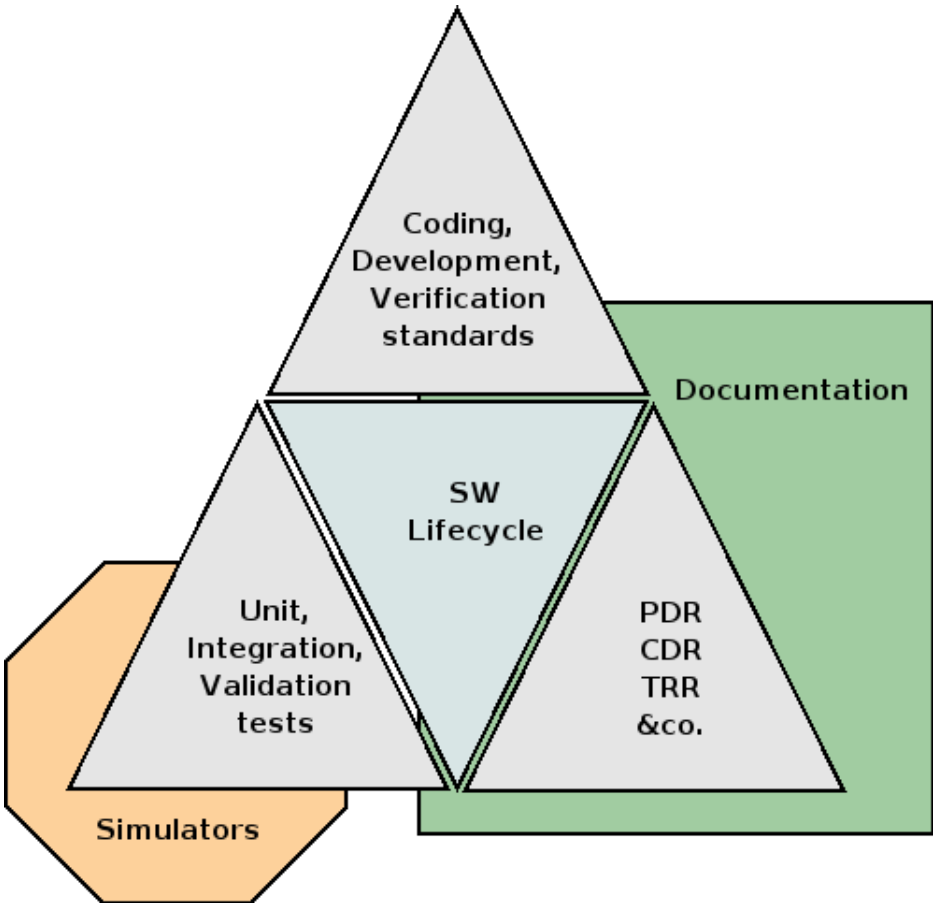
$\Downarrow$

Since we have plenty (1Mb) of space, we can afford a COTS realtime kernel: RTEMS.





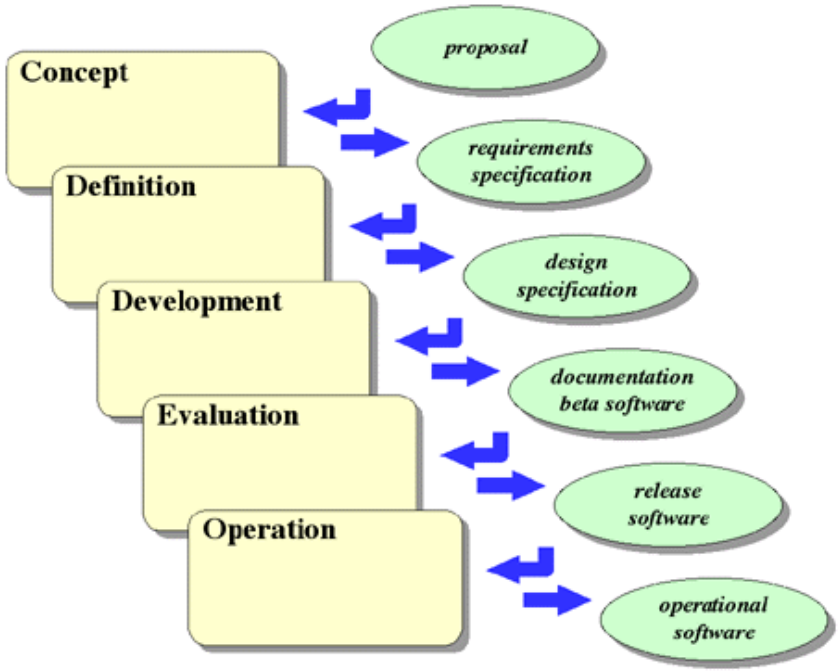
Quality matters

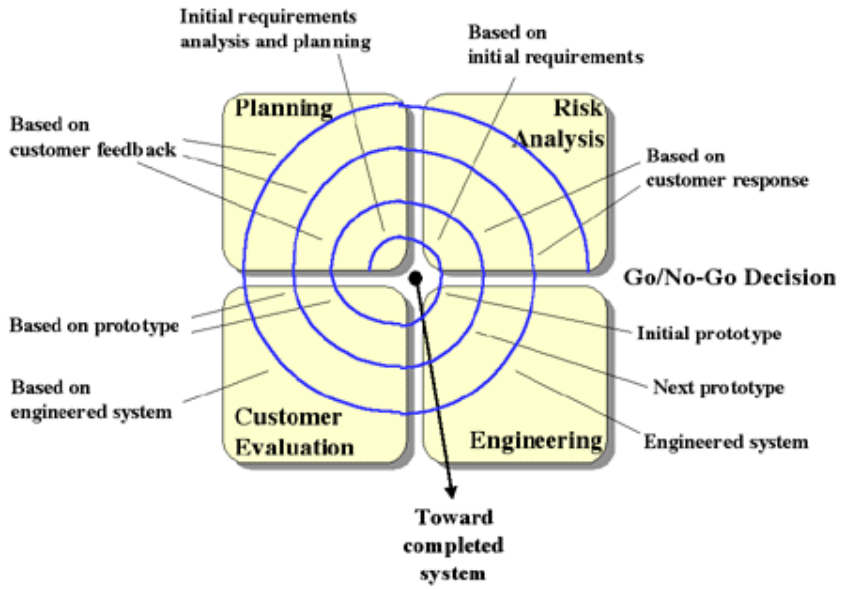


Quality Assurance is an integral process encompassing all stages of the software development life-cycle.

- It relies on a series of engineering standards, taylored from ESA templates, and covering all key aspects of the development.
- Testing is central. Defined in the verification standards, includes thorough unit testing (with processor emulators support) during the coding phase, followed by (module) integration tests (using simulators for all external subsystems communicating with the DMU) and, finally, validation test against the mission requirements.
- A well defined engineering process is in place, relying on key milestones (PDR, CDR, TRR) with the participation of all stakeholders.
- All the process is captured in a solid, peer-reviewed documentation set, mimicking the waterfall development model: System Requirements, Software Requirements, Architectural and Detailed Design, Validation and Verification plans.

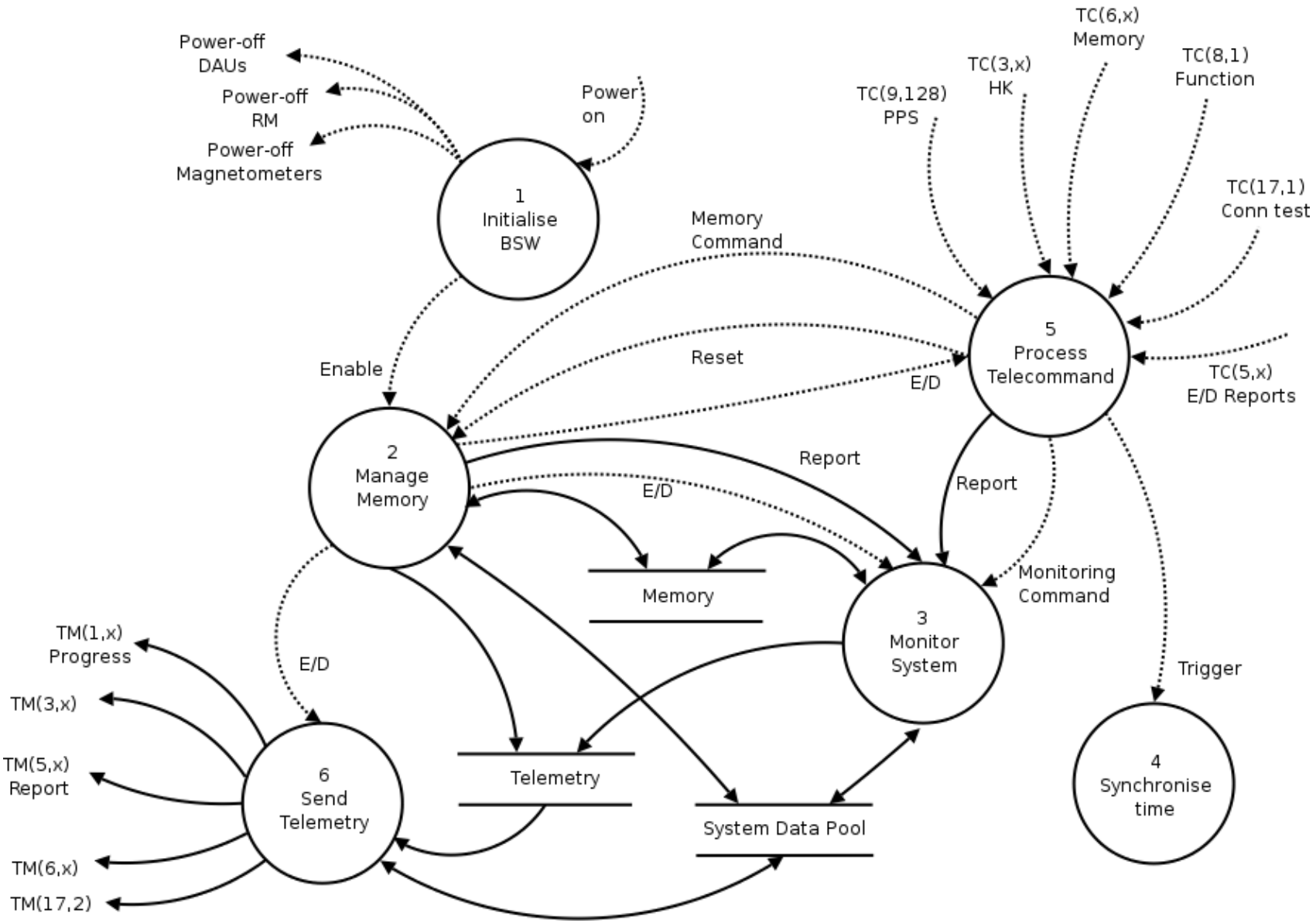
The software life-cycle (on paper and for real)





The ideal waterfall process (left) is the expected outcome of ESA's Engineering Standards. It works as a reporting template for the final product, whose actual development is much closer to the well-known *spiral model* (right). Especially noteworthy is (unit) testing, which actually drives code writing, in the TDD spirit.

Ward-Mellor structured design



Tools galore: taming code complexity

- Automatic tools
  - Code coverage (*Cantata*). 100% execution and branch.
  - Code linting (*splint*).
  - Metrics extraction: MacCabee, cyclomatic (*Cantata*).
- Semi-automatic tools
  - In-code documentation (*doxygen*).
  - Test libraries, stubbing and drivers (*Cantata*).
  - Software and documentation distributed version control system (*GNU Arch*).
- Plain tools
  - GNU cross-compilation tool chain.
  - ERC32 emulator.
  - LaTeX, a programmable documentation system.
  - *MoinMoin WikiWiki*, a centralised information repository.
  - *Python*: gluing it all together.